Gaston

A plotting utility for Julia

v. 0.5.4

M. Bazdresch

June 29, 2012

Please note: Gaston is currently under heavy development, and all functions and definitions are subject to change from one version to the next, as we figure out the best way to organize the code. Gaston has been tested on Linux (Ubuntu 10.04 and Arch), with gnuplot 4.6, and Julia 38ec7cfa0d (built on 2012-06-26).

Contents

1 Introduction

Gaston provides a way to plot scientific and numeric data using the Julia programming language. It accomplishes this by harnessing gnuplot, a versatile and time-tested plotting utility. Gaston also relies on gnuplot for interacting with plots (zooming and rotating a plot with the mouse, for instance).

The primary purpose of Gaston is to provide easy-to-use functions to quickly and conveniently plot the most common kinds of scientific and numeric data. It is concerned mainly with screen output, although it also supports exporting figures to SVG, PDF, PNG and GIF files (see section 6 for more on printing).

Gaston offers two sets of plotting functions. The first, called "high-level", is similar to some of the functions offered by programs such as Octave. These functions may be used to create 2-D plots (including histograms), 3-D surfaces, and for plotting images. See section 4 for more information on these functions.

The second set, called "mid-level", offers more flexibility at the cost of being a bit more cumbersome. With these functions, figures are built step by step by adding sets of coordinates with associated properties. These functions allow creation of some kinds of plots not supported by the high-level interface. For example, plotting a histogram and a curve on the same plot is only possible with mid-level functions. See section 5 for more information.

At the lowest level, Gaston provides a function called `gnuplot_send()`. It takes a string argument, which is sent to a gnuplot process through a pipe. While certainly not user-friendly, this function may be used to ultimately access any gnuplot feature from Julia, even if not supported by Gaston.

Gaston has a layered design: high-level plot functions use the mid-level functions to build and specify plots, which are translated to gnuplot commands and sent to gnuplot using `gnuplot_send()`.

2 Installation

To use Gaston, follow this procedure:

1. Save all files `gaston*.jl` somewhere convenient. Then, you may `cd` to that directory and start julia there, or do

   ```
   push(LOAD_PATH, "/path/to/gaston/jl")
   ```

   Then, load the program with

   ```
   load("gaston.jl")
   ```

2. To run a demo, do

```
load("gaston_demo.jl")
demo()
```

This will create a series of figures that illustrate the current capabilities of the program. The same file may also serve as a guide on how to create different types of plots.

3. To run the tests (see section 9), do

```
load("gaston_test.jl")
run_tests()
```

All tests should pass.

3 Definitions

A **figure** is an independent window, which contains a set of axes, on which one or more **curves** are plotted. A figure may contain **labels** (for instance, on each coordinate axis), a **title**, and a **legend box** which identifies each curve. Gaston supports having any number of figures open at the same time; however, gnuplot requires that only one figure is able to offer mouse interactivity at a given time. Each figure is identified by a unique **handle**. Handles are natural numbers.

A **curve** is defined by a set of coordinates. Two-dimensional curves have **x** and **y** coordinates; in three dimensions, an additional **z** coordinate must be specified. A curve also has several **properties** that specify a plotting configuration (for instance, it may have a **plotstyle**, a **linewidth**, a **linecolor**, etc), which define how the curve is to be plotted.

4 Plotting

4.1 Figures

A figure may be created by the function `figure()`. Many plotting functions create or reuse an existing figure, as needed (see each function's documentation). Called with no arguments, `figure()` creates a new figure with the smallest available handle. Called with an argument (which must be a natural number), it will create a figure with that handle if no such figure exists, or will select it (make it current) if it exists.

Selecting an existing figure may be useful, for instance, to interact with it using mouse and keyboard, or to overwrite it with the next plotting command.

Handles may be created in any numerical order.

This function always returns the handle of the current figure.

4.2 Terminals

Gaston supports plotting to the screen as well as printing the plots to files. Three screen terminals are supported: `wxt`, `x11` and `aqua`[1]. The `x11` terminal is faster than `wxt`, but it offers lower plot quality; it is provided for use in systems with lower performance or that don't support WxWindows.

For printing to files, terminals `svg`, `pdf`, `png`, `eps` and `gif` are supported. For more details on printing, see section 6 below.

To set the terminal type, use the command

```
set_terminal(term)
```

where `term` is a string.

4.3 2-D plotting

There are two commands for two-dimensional plotting: `plot()` and `histogram()`.

4.3.1 plot()

The `plot()` function takes at least one argument, with the following format:

```
plot({h,} {x,} y {, property, value...} {...})
```

- If the optional argument `h` is provided, it is assumed to be the figure handle in which to plot.
  - If the handle doesn't exist, a new figure is created.
  - If it exists, the figure will be overwritten.
  - If it is `0`, then a new figure to plot in will be created, using the next handle available.

  If it is not provided, then the current figure will be used and overwritten.
- The `x` and `y` arguments specify coordinates (they must be ranges, vectors, or two-dimensional arrays with a singleton dimension). If only `y` is provided, it is assumed to be the ordinate. If `x` and `y` are provided, they are assumed to be the abscissa and the ordinate, in that order.
- The coordinates may be followed by any number of `property`, `value` pairs. These are used to set the value of any of the curve's or the axes' properties (see section Reference, below). `property`s are always strings. The `value`s must be of the appropriate type.

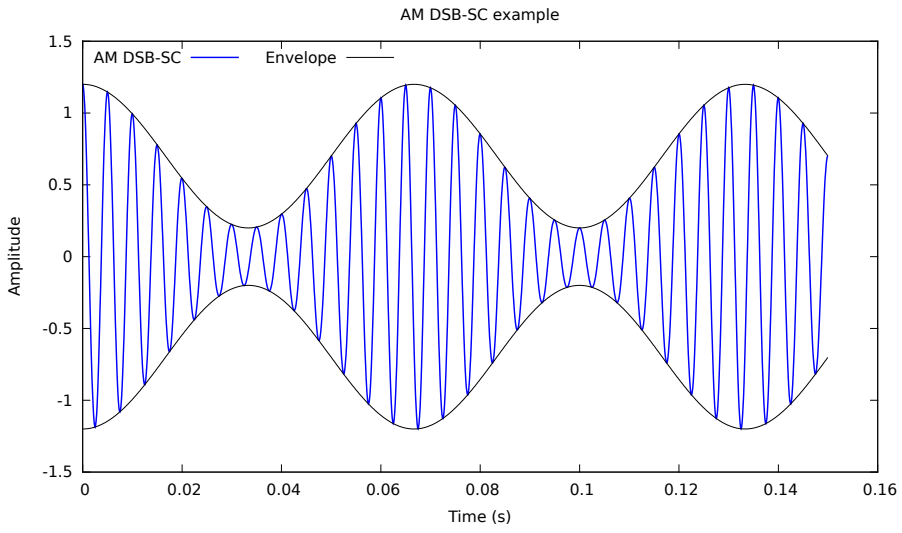
[1] Gnuplot provides the `aqua` terminal on Mac OS X systems only. It is included in Gaston for convenience of users of that system. However, the author does not have access to any Mac computers and, in consequence, this terminal should be considered as unsupported. Bug reports, patches and comments are welcome, as always.

- The pattern `{x,} y {, property, value...}` may be repeated any number of times.
  - Curve settings are always set for the immediately preceding curve.
  - Axes settings may be specified at any time, and in the case of repeated properties, the last one set is the one that is used.
- Properties may take the following values: `plotstyle`, `legend`, `color`, `marker`, `linewidth`, `pointsize`, `title`, `xlabel`, `ylabel`, `box`, `axis`.
- Supported plotstyles are `lines`, `linespoints`, `points`, `impulses`, `dots` and `boxes`.
- plot() returns the handle of the figure that was plotted.

As an example, to plot an amplitude modulated signal and its envelope, we may run the following code:

```
t = 0:0.0001:.15
carrier = cos(2pi*t*200)
modulator = 0.7+0.5*cos(2pi*t*15)
am = carrier.*modulator
plot(t,am,"color","blue","legend","AM DSB-SC","linewidth",1.5,
    t,modulator,"color","black","legend","Envelope",
    t,-modulator,"color","black","title","AM DSB-SC example",
    "xlabel","Time (s)","ylabel","Amplitude",
    "box","horizontal top left")
```

which produces this plot:



4.3.2 histogram()

The `histogram()` function plots a single histogram in a figure. It has the following format:

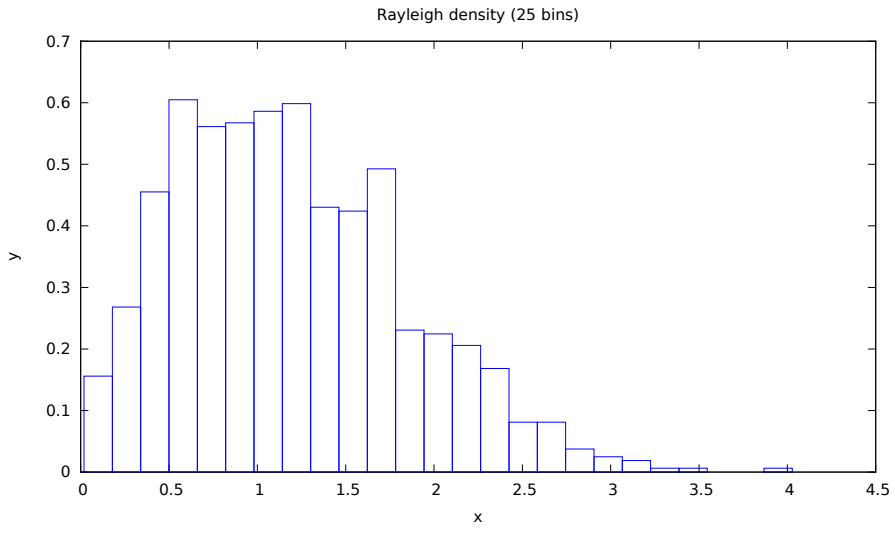
```
histogram({h,} y {, "bins", bins} {, "norm", norm} {, property, value...})
```

- The optional argument `h` has the same meaning as in `plot()`.
- The histogram consists of boxes, where the height of each box is given by the number of elements of `y` that fall in a given range.
- `bins` specify the number of bins (boxes) that will be plotted (`bins` must be an integer larger than zero).
- If `norm` is specified, the histogram will be normalized, so that the area under the histogram is equal to `norm` (`norm` must be a real number larger than zero).
- The pairs `{, property, value...}` have the same meaning as in `plot()`.
- Properties may take the following values: `legend`, `color`, `linewidth`, `title`, `xlabel`, `ylabel`, `box`.
- `histogram()` returns the handle of the figure that was plotted.

As an example, to plot an approximation of a Rayleigh density, we may run the following code:

```
y = sqrt( randn(1000).^2 + randn(1000).^2 )
histogram(y,"bins",25,"norm",1,"color","blue","title","Rayleigh density (25 bins)")
```

which produces this plot:



4.4 3-D plotting

There is one 3-D plotting command, `surf()`. This command is intended for plotting surfaces. Its format is as follows:

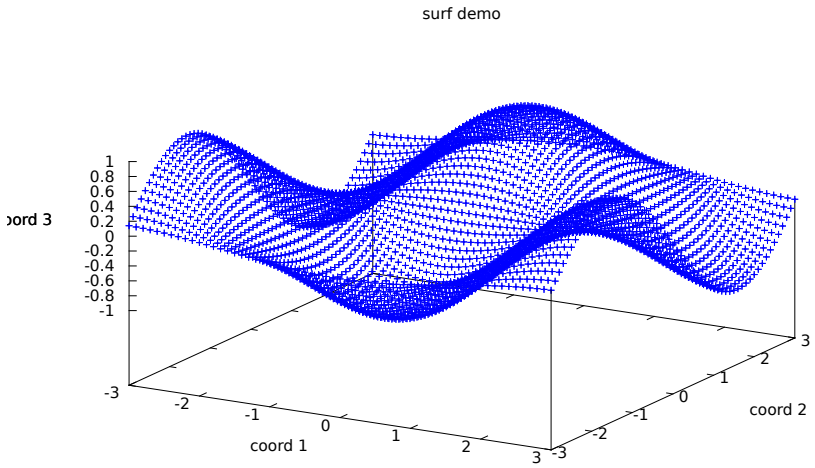
```
surf({h,} {y, {x,}} {f|Z} {, property, value...} {...})
```

- The optional argument `h` has the same meaning as in `plot()`.
- The `x` and `y` arguments have the same meaning as in `plot()`.
- The argument `f|Z` may be either a 2-D matrix `Z` or a function of `(x,y)`. If `Z` is provided, then its element `i,j` is assumed to be the z-coordinate associated with `(x[i],y[j])`. If `f` is provided, then `x` and `y` must be provided too, and the function `meshgrid()` will be used to calculate matrix `Z`.
- The pairs `{, property, value...}` have the same meaning as in `plot()`.
- Properties may take the following values: `plotstyle`, `legend`, `color`, `marker`, `linewidth`, `pointsize`, `title`, `xlabel`, `ylabel`, `zlabel`, `box`.
- Supported plotstyles are `lines`, `linespoints`, `points`, `impulses`, `dots` and `pm3d`.

As an example, to plot `f(x,y)=cos(x)*sin(y)` using points, one may use the following code:

```
gnuplot_send("set view 67,25")
surf(-3:.1:3,-3:0.1:3,(x,y)->cos(x)*sin(y),"plotstyle","points",
"xlabel","coord 1","ylabel","coord 2","zlabel","coord 3",
"title","surf demo","color","blue")
```

which produces this plot:



Note the use of `gnuplot_send()` to set the view, which is something Gaston doesn't support natively yet.

4.5 Plotting images

The command to plot images is `imagesc()`. This command plots a matrix as an image. The format of the arguments is the following:

```
imagesc({h,} {y, {x,}} Z {, "clim", clim} {, property, value...})
```

- The optional argument `h` has the same meaning as in `plot()`.
- The `x` and `y` arguments have the same meaning as in `plot()`.
- `Z` is a matrix. If it is two-dimensional, then gnuplot's `image` plot style is used. In this case, gnuplot automatically adjusts the values in `Z` to fit the current palette. If it is three-dimensional, then the plotstyle `rgbimage` is used; in this case, all matrix elements are assumed to be between 0 and 255. `Z[:,:,1]` is assumed to be the image's red component, `Z[:,:,2]` the blue, and `Z[:,:,3]` the green.
- If the optional `clim` property is specified, then it must be followed by a two-element array `clim=[cmin cmax]`. The values of Z are scaled to the range `0:255` using the following logic:

```
Z -= cmin
Z[Z<0] = 0
Z *= 255/(cmax-cmin)
Z[Z>255] = 255
```

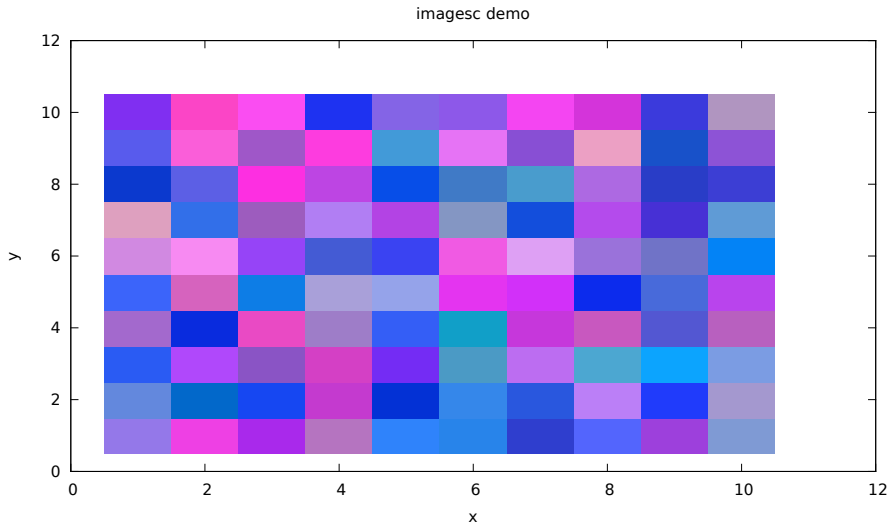
- The pairs `{, property, value...}` have the same meaning as in `plot()`.
- Properties may take the following values: `title`, `xlabel`, `ylabel`.
- `imagesc()` returns the handle of the figure that was plotted.

It is important to understand how `imagesc()` interprets matrix `Z`. The matrix is plotted such that the resulting image resembles the matrix as it would normally be printed on screen. That is, element `Z[1,1]` (or `Z[1,1,:]` for an RGB image) is plotted on the upper left-hand corner, and matrix columns are plotted as image columns.

As an example, the following code:

```
z = zeros(10,10,3)
z[:,:,1] = 255*rand(10,10)
z[:,:,2] = 128*rand(10,10)+40
z[:,:,3] = 64*rand(10,10)+190
imagesc(z,"title","imagesc demo")
```

produces a plot similar to this:



5 Plotting with mid-level functions

In addition to the plotting functions described in the section above, Gaston offers a "mid-level" set of functions that allow plots to be created step-by-step.

Having this mid-level layer has two benefits. One is that this layer is more flexible and allows direct control over all aspects of the plot. For instance, using high-level functions it

is not currently possible to plot more than one histogram, or a histogram and another curve, on the same figure. The mid-level layer allows such combinations. Another example is an algorithm that builds figures step-by-step as data becomes available, instead of waiting until all the data needed has been produced.

A second benefit is that this layer abstracts Gaston's internal graphics representation from the high-level layer. This means Gaston's whole back-end may change without affecting the high-level functions; only the mid-level layer would have to be adapted.

Also, much of Gaston's error checking and argument validation is performed in this layer.

Please note that there is a single mid-level plot command, which is `llplot()`. According to the type of coordinates and plotstyle, it will figure out how to plot a figure.

5.1 2-D plotting

Plotting proceeds in steps:

1. Create or select a figure with `figure()` or `figure(i)`, where `i` is a positive integer.
2. Add a curve (a set of coordinates plus a plot configuration), with `addcoords(x,y,conf)`. Here, `x` and `y` are vectors, and `conf` configures the plot and line styles, markers, legend, color, etc. Repeat this step for each curve you wish to include in the figure.
3. Add a configuration for the entire figure (axis), with `addconf(conf)`, where `conf` contains the figure configuration.
4. Issue the `llplot()` command.

A curve configuration is created as follows:

1. Create a default configuration with `c = CurveConf()`.
2. `c` is a structure, each of whose fields controls one aspect of the curve's configuration. These fields may be set individually. Available fields are: `legend`, `plotstyle`, `color`, `marker`, `linewidth`, and `pointsize`. For instance, to change a curve's color, do[2]

   ```
   c.color = "blue"
   ```

   See gnuplot's documentation and this manual's section 7 to see the range of valid options.

A figure (axis) configuration is created as follows:

1. Create a default configuration with `a = AxesConf()`

[2]Directly changing a structure's fields, instead of using setter functions, may be seen as non-idiomatic or inelegant. In this case we have decided to do the simplest thing that might possibly work.

2. Just as in the case of a curve configuration, `a` is a structure whose fields may be modified. Available fields are: `title`, `xlabel`, `ylabel`, `zlabel`, `box`, `axis`.

Several rules apply:

- You can create as many figures, each with as many curves, as desired.
- Whenever possible, missing coordinate data will be inferred. For example, calling `addcoords()` with a single vector `y` will assume the `x` coordinate is `1:length(y)` and set up the default plot configuration.
- If you call `addcoords()` with matrix arguments, each column will be interpreted as a different plot.
- Calling `addcoords()` will create a new figure if none have been created yet.
- Calling `llplot()` without an axis configuration will just use one by default.
- Gnuplot only provides mouse interaction support for the current figure. To use the mouse in a previously created figure `i`, just issue command `figure(i)`. This will also bring the figure to the front.

5.1.1 2-D plotting styles

Besides simply plotting a set of `x` and `y` coordinates, Gaston supports other kinds of plots.

Error bars and lines. Gaston supports gnuplot's `errorbars` and `errorlines` plot styles. To add error bars or lines, follow `addcoords()` with a call to `adderror()` with one or two extra coordinates. If two coordinates are provided, they are interpreted as `ylow` and `yhigh` (following gnuplot's terminology). If only one is provided, it is interpreted as `ydelta`. Remember to configure the plotstyle accordingly.

Histograms. To plot the histogram of a data vector `data` with `b` bins, first use the auxiliary function `(x,y) = histdata(data,b)` to create `x` and `y` coordinates that may be plotted with the `boxes` plotstyle.

5.2 3-D plotting

The same rules apply, except that `addcoords()` should be called as `addcoords(x,y,Z)`, where `Z` is a matrix whose element `j,k` corresponds to some function of `x[j],y[k]`.

For convenience, a function `Z = meshgrid(x,y,f)` is provided. Called with `x`, `y` coordinates and a function `f`, it will return a matrix that may be used to plot `f`.

5.3 Image plotting

Two image plotstyles are supported: `image` and `rbgimage`. Plotting images is very similar to 3-D plotting, the only differences being the plotstyle and the way the coordinate data is interpreted.

Scalar image. To plot a 2-D matrix `Z` as an image, set the plotstyle to `image`. The color of each image element is given by the matrix values and the current colormap.

RGB image. To plot an RGB image, the matrix `Z` must be three dimensional, and the plotstyle must be set to `rgbimage`. `Z[:,:,1]` is assumed to be the image's red component, `Z[:,:,2]` the blue, and `Z[:,:,3]` the green.

6 Printing to a file

Gaston supports plotting a figure to a file instead of the screen (*printing* a figure). Currently supported are PDF, PNG, SVG and GIF files. There are two ways to print figures.

6.1 Printing a single figure

If you have a figure on screen that you want to print, you may use the `printfigure()` command. The following example shows hows to print a figure to a PNG file:

```
plot(sin(-pi:0.01:pi))
set_filename("test.png")
printfigure("png")
```

If the filename is not set, output will be sent to the screen.

Some properties of the printed figures can be controlled using the following functions:

`set_print_color()`: Use `"mono"` for printing monochromatic figures, and `"color"` for color figures.

`set_print_fontface()`: Name of font to use for all text in the figure. Use `fc-list` to find fonts installed on your system.

`set_print_fontsize()`: Font size.

`set_print_fontscale()`: Gnuplot scales the fonts by this factor (even if the font size has been specified).

`set_print_linewidth()`: Controls the plot's linewidth. This is useful because in most cases, gnuplot's default linewidth is too thin.

`set_print_size()`: Sets the plot size. The size is specified differently for various terminals, see table below.

The following table provides more details on each function.

Command	Values	Default	File types
`set_print_color()`	`"mono"`, `"color"`	`"color"`	PDF, PNG
`set_print_fontface()`	Font name (a string); font must be present in the system	`Sans`	All
`set_print_fontsize`	Positive number	12	All
`set_print_fontscale`	Positive number	0.5	PDF, PNG, GIF
`set_print_linewidth`	Positive number	1	All
`set_print_size()`	For GIF, SVG, PNG, EPS and PDF: `"X,Y"` specifies a plot of `X` times `Y` pixels. In addition, for PNG, EPS and PDF, dimensions may be specified as `"Xin,Yin"` or `"Xcm,Ycm"`	`"5in,3in"`	All

6.2 Always print to files

If you want regular plot commands to print to files instead of showing the plots on the screen, just set the terminal type and filename at the start of your session. For example, Gaston may be used in a webserver that reads SVG plots from a pipe; this may be set up as follows:

```
set_terminal("svg")
set_filename("|serverpipe")
plot(sin(-3:0.01:3),"title","SVG test")
```

The plot command in the last line will cause the SVG data representing the figure to be sent to the pipe `serverpipe`.

7 Reference

Note: In this section, at least superficial knowledge of Julia and gnuplot is assumed. Please refer to the respective documentation for more details.

7.1 Curve configuration

A given curve's configuration is stored in a structure of type `CurveConf`. This structure has the following fields:

Field	Notes	Meaning in gnuplot
`legend`	The text that appears in the legend box.	Argument of `title`.
`plotstyle`	How the curve will be plotted.	Argument of `with`.
`color`	The curve color.	Argument of `linecolor rgb`.
`marker`	The marker name.	Argument of `pointtype`.
`linewidth`	The curve line width; must be a positive number.	Argument of `linewidth`.
`pointsize`	The marker size; must be a positive number.	Argument of `pointsize`.

The types, default and valid values for each field are given in the following table.

Field	Type	Default value	Valid values
`legend`	`String`	`""`	Any string.
`plotstyle`	`String`	`lines`	`lines`, `linespoints`, `points`, `impulses`, `errorbars`, `errorlines`, `pm3d`, `boxes`, `image`, `rgbimage`.
`color`	`String`	`""`	`""`, any gnuplot color name, or a color specified in a string in the format `"#RRGGBB"`.
`marker`	`String`	`""`	`""`, `+`, `x`, `*`, `esquare`, `fsquare`, `ecircle`, `fcircle`, `etrianup`, `ftrianup`, `etriandn`, `ftriandn`, `edmd`, `fdmd` (run `test` in gnuplot terminal).
`linewidth`	`Real`	1	Any real number
`pointsize`	`Real`	0.5	Any real number

Notes: When `color` or `marker` are set to the empty string, gnuplot will use its own default values. Gaston does not verify that the color name provided is valid. You can see a list of valid color names running `show colornames` in a gnuplot terminal.

The following table lists the marker names and corresponding symbols.

Marker name	Symbol
`+`	+
`x`	×
`*`	∗
`esquare`	□
`fsquare`	■
`ecircle`	○
`fcircle`	●
`etrianup`	△
`ftrianup`	▲
`etriandn`	▽
`ftriandn`	▼
`edmd`	◇
`fdmd`	◆

7.2 Axes configuration

There may be only one configuration per figure. This configuration is stored in a structure of type `AxesConf`, which has the following fields:

Field	Notes	Meaning in gnuplot
`title`	The figure's title	`title-spec`
`xlabel`	Abscissa's label	Argument to `set xlabel`
`ylabel`	Ordinate's label	Argument to `set ylabel`
`zlabel`	Z-axis label	Argument to `set zlabel`
`box`	Legend box configuration	Argument to `set key`
`axis`	Axis scale	Argument to `set logscale`

The types, default and valid values for each field are given in the following table.

Field	Type	Default value	Valid values
`title`	`String`	`"Untitled"`	Any string
`xlabel`	`String`	`"x"`	Any string
`ylabel`	`String`	`"y"`	Any string
`zlabel`	`String`	`"z"`	Any string
`box`	`String`	`"inside vertical right top"`	Any valid string
`axis`	`String`	`""`	`""`, `normal`, `semilogx`, `semilogy`, `loglog`

Notes: Gaston does not verify that `box` contains a valid `set key` argument. The `axis` values follow Matlab's conventions for logscale axes; Gaston translates them to gnuplot's equivalents.

7.3 Changing default configuration

Gaston provides functions to change the default values listed above. There is one function per each property, both for curves and for axes. For example, a Spanish speaker may wish to change the default value for figure titles from "Untitled" to "Sin título". This is achieved by

```
set_default_title("Sin título")
```

and, from that point on, any figure with an unspecified title will be titled thus.

Functions to change the defaults have the form `set_default_*(arg)`, where `*` is the property name and `arg` is the new default value (must be of the appropriate type).

7.4 Plot types

In this section, we describe how Gaston decides which kind of plot to produce (2-d, 3-d, or image), based on the available coordinates and specified plot style.

In the following table, a check mark (✓) means that the coordinate has been specified by the user in either a mid-level or high-level command (`x` coordinates, if not specified, are calculated by Gaston, and are not included in the following table. The same applies to `yhigh`).

plotstyle	y	Z	ylow	Gnuplot command
`lines`	✓			`plot with lines`
	✓	✓		`splot with lines`
`linespoints`	✓			`plot with linespoints`
	✓	✓		`splot with linespoints`
`points`	✓			`plot with points`
	✓	✓		`splot with points`
`impulses`	✓			`plot with impulses`
	✓	✓		`splot with impulses`
`dots`	✓			`plot with dots`
	✓	✓		`splot with dots`
`boxes`	✓			`plot with boxes`
`errorbars`	✓		✓	`plot with errorbars`
`errorlines`	✓		✓	`plot with errorlines`
`pm3d`		✓		`splot with pm3d`
	✓	✓		`splot with pm3d`
`image`		✓		`plot with image`
	✓	✓		`plot with image`
`rgbimage`		✓		`plot with rgbimage`
	✓	✓		`plot with rgbimage`

Any other combination of coordinates and plotstyle produces undefined behavior – we try to identify invalid combinations and produce an error, but this is not guaranteed. Please note that mixing 2-d and 3-d plots on the same figure also produces undefined behavior.

7.5 Global variables

Gaston introduces two global variables to the namespace. All global variables are defined in file `gaston.jl`.

Name	Type	Purpose
gnuplot_state	GnuplotState	Store the state of the current gnuplot process and information needed to interact with it.
gaston_config	GastonConfig	Stores Gaston's configuration.

7.6 Types

Gaston defines several new types. All types are defined in file `gaston_types.jl`.

Name	Purpose
GnuplotState	Structure that stores the state of gnuplot process.
GastonConfig	Structure that stores Gaston's configuration.
CurveConf	Configuration of a single curve.
AxesConf	Configuration of a figure.
CurveData	A set of coordinates and its configuration.
Figure	Structure that stores a set of curves, a configuration and a handle.
Coord	A variable of this type may be used as an abscissa or ordinate.

8 Notes to the user

This section gathers some notes that may be useful to Gaston users and don't fit anywhere else on this document.

8.1 Scripting

When including Gaston plotting commands in a script, it is necessary to take steps to avoid race conditions between Gaston and Gnuplot. These race conditions arise for the following reasons.

Gaston writes the data to be plotted to a file, and then issues plot commands that point Gnuplot to the correct file. However, communication between Gaston and Gnuplot occurs asynchronously. This means that, when Gaston communicates a command to Gnuplot, it writes the command to a pipe and then continues with its normal execution. If two consecutive (or near) plot commands write plot data to the same file, it is possible that the file will be overwritten while Gnuplot was still reading it.

When this happens, the plot command will apparently succeed, but no plot will be produced, and Gnuplot will issue error messages that appear on the screen. If this happens, it may be necessary to insert `sleep(x)` commands between the plotting commands. Here, `x` is

the number of seconds to wait; this number will change depending on the particular system's speed. A value of 0.1 could be more than enough on modern machines.

A solution to this problem is under investigation.

9 Testing

Gaston includes a unit test framework that verifies the program's functionality. To use it, issue

```
load("gaston_test.jl")
```

To run all tests, run:

```
run_tests()
```

This function will run all tests and print a summary to the screen.

There is a peculiarity to the tests that must be taken into account. When plotting a figure, Gaston will write the relevant data to a file and then issue commands to gnuplot through a pipe. These two actions are sequential but asynchronous. When plotting many figures in quick succession, it may happen that a data file is updated before gnuplot has processed it. In this case, the commands that gnuplot receives are no longer correlated with the data file, and the commands may fail.

When running tests, this is exactly what may happen. This is prevented by inserting a delay between tests, using a system call to `sleep` (see macros `test_success` and `test_error` in file `gaston_test.jl`). The sleep amounts may have to be increased for more complex tests or for slower computers. A more robust solution may be implemented in future versions of Gaston.

9.1 Adding tests

New tests are easy to add. To test code that should complete successfully, use the macro `@test_success`, as in this example:

```
@test_success plot(-3:3,"title","test")
```

This code should be added to function `run_tests_success()`. To test code that should produce an error, use the macro `@test_error` inside function `run_tests_error()`.